

2.0 Verification Results

The results of verification of five FEs are provided in this section of ASP II. The FEs include Signature Fluctuations, Multipath/Diffraction, Threshold, MTI, and Signal Integration. Significant findings and implications for model use for each FE are summarized below.

Signature Fluctuations: Errors in checking theoretical limits on P_{fa} and P_d were found in subroutine RDRERR, a minor error was found in subroutine RCSPRT, and overflow errors can occur in subroutine THRESH. In addition, the developer should consider changing the calculation of "loss" for a non-fluctuating target.

Code quality is generally good; however, the design makes tracing fluctuation effects into the radar range equation somewhat difficult. Additional comments are recommended.

Internal documentation is fairly adequate, but could be improved. Headers for the subroutines GETRCS, RCSINP, and RCSINT omit some standard information. Additional comments are recommended to emphasize or explain fluctuations aspects of several subroutines. The most serious deficiency is the misinformation about the log-normal distribution.

The external documentation for ALARM 3.0 is inadequate. The most serious problem is in the User's Manual, which gives incorrect directions for the fluctuation table inputs and gives incorrect bounds on the variables PSUBD and PSUBFA. According to Blake [A.1-4], they should be $0.1 \leq \text{PSUBD} \leq 0.9$ and $10^{-12} \leq \text{PSUBFA} \leq 10^{-4}$. In addition, the User's Manual should clearly explain bounds on the fluctuations table. It also should emphasize that the azimuth bound is for the total number of azimuth segments, not just half when the symmetry flag is true.

The descriptions of subroutines THRESH and RCSINT in the Programmer's Manual do not mention the fluctuations portion of these routines; descriptions of these portions should be added. The Analyst's Manual gives a good description of fluctuations in the section on detection theory; a reference to this in the section on target RCS might be helpful.

Multipath/Diffraction: The broadest discrepancy is that ALARM 3.0 does not exactly match the MIT Lincoln Laboratory SEKE code; the following three items differ: (1) the constants in the logic to determine which propagation effects will be calculated, (2) the definition of the terrain profile, and (3) the extent of the tangent plane used in the calculation of the multipath effect. Apart from the differences from SEKE, several minor discrepancies were found in subroutines MLTPTH, VISBLE and SEKINT. The ALARM external documentation for this FE is inadequate, because the analyst's manual gives an unacceptably incomplete description of the relevant algorithms.

Threshold: One code discrepancy was found in the preliminary comparisons of S/I to threshold (T) in subroutines PULSED and PULDOP. The error occurs when the preliminary S/I equals T, so the comparison test is not passed and additional sources of interference are (generally) not computed.

A second discrepancy is the error in checking the theoretical limits on P_{fa} and P_d in RDRERR; this error is due to using referenced algorithms (from Blake, [A.1-4]) without maintaining the limits specified by the reference.

A third possible problem is not a discrepancy, since the code implements the design as described in section 3.3.7 of the User's Manual, and as mentioned in the Programmer's Manual (pages 22-23 and 25-26) and the Analyst's Manual (section 4.2.6). However, it seems inconsistent that no distinction is made between target detection and deceptive jammer detection in contour plot mode, but this distinction is made in flight path mode. A re-examination of this design issue is recommended to the model developer.

Code quality is generally very good in ALARM proper, but not in the post-processing program BINPRO. The major problem in BINPRO is that it seems to have been written for a UNIX system and then modified to run on VMS. This can lead to confusion for VMS users.

Internal documentation in ALARM is adequate, but should be improved to provide a consistent set of information in the subroutine headers.

Several discrepancies were found in the ALARM manuals.

The User's Manual contains several errors in the section describing the binary plot file (page B-3). In addition, the Input Guide in the User's Manual gives incorrect bounds on PSUBD and PSUBFA. Appendix F of the User's Manual describes the support programs for ALARM. BINPRO is discussed on pages F-14 through F-18. The inputs are not described in the same detail as for the ALARM model proper.

The Programmer's Manual states that program DETECT is described in appendix F of the SUM, but DETECT is no longer described there. Descriptions of BINPRO, PREPGP, and GNUPLOT should be added to the Programmer's Manual.

The Analyst's Manual gives a very good description of the calculation of the threshold value in section 4.4.6, but the last paragraph on page 61 could be mistakenly interpreted as implying that ALARM provides default values of P_d and P_{fa} . This manual also provides a very good description of the calculation of S/I in section 4.2.6, but it should note (as a unique pulse doppler aspect) that the S/I over all PRFs is selected as the overall S/I for the radar.

MTI: No major problems or anomalies were found with the ALARM MTI references or the code implementation. The minor problems found in RDRERR could be eliminated by replacing lines 522 and 560 with the following:

Line 522: 'IF (RMTIMX(IGATE) .LT. RMTIMN(IGATE)) THEN '

Line 560: 'IF (AMTIMX(IGATE) .LT. AMTIMN(IGATE)) THEN'

Code quality is generally good. Internal documentation is adequate, but should be improved to provide a consistent set of information in the subroutine headers and to correct spelling errors.

External documentation for MTI in ALARM 3.0 is generally good. The manuals have been updated to match the new code in this version. The minor error in the User's Manual could be corrected by replacing "NDELAY > 0" with "NDELAY 0".

Integration: No major errors were found. Errors in checking theoretical limits on P_{fa} and P_d were found in RDRERR; these errors are due to using referenced algorithms without maintaining the limits specified by the reference.

Code quality is generally good, although there are some unused input variables with minor discrepancies between the printed output from RDRPRT and the ALARM input guide. Internal documentation is adequate, but should be improved to provide a consistent set of information in the subroutine headers.

The external documentation is somewhat inadequate. The Input Guide in the User's Manual gives incorrect bounds on PSUBD and PSUBFA. In addition, the Input Guide should warn users that the value entered for NPULSE should be the equivalent number of pulses integrated and refer to table 2.2 in Blake [A.1-4]. (The Analyst's Manual warns users to carefully define the number of pulses integrated, but it does not give specifics or mention equivalent number of pulses integrated.) Also, the Input Guide should refer to parameter names (not specific values) for array dimensions.

2.0.1 Code Verification Methodology

The SMART verification process differs from classical verification in two main areas. First, the SMART process is structured according to the FAT; i.e., each FE is verified as a unit. This contrasts with the classical verification process structure which is based primarily on the software call hierarchy. The second primary difference is that the classical process depends on software requirements and design documents written before the code was developed, while the SMART process addresses mature models for which these documents do not exist. Thus, the SMART

verification process is carried out in three major stages: (1) creating a post-development design document, referred to as the Conceptual Model Specification (CMS); (2) desk checking; and (3) software testing.

CMS Development

The purpose of the CMS is to describe the developer's conceptual design and specifications for the model, so that verification using the Military Operations Research Society (MORS) definitions, described in section 2.0.2. below, can be performed. The DOD-STD-2167A design document standards were reviewed to develop a format tailored for utility in verifying mature models. The CMS is currently being written; to date, several FEs are included. ASP II, section 2, contains the CMS for these FEs.

The CMS contains descriptions of operational concept, top-level design, and detailed design. For each FE, the detailed design consists of a brief theoretical description, broad design requirements, engineering design approach, software design, and assumptions and limitations.

Desk Checking

Desk checking is a manual process that consists of three types of procedures: (1) correlating design with cited references, (2) correlating code with design (logical verification), and (3) code auditing (code verification). Note that verification of intelligence data is performed by government agencies and is not part of this effort.

The first procedure is done simply by examining the CMS design descriptions and verifying that the described equations and algorithms accurately represent or can be derived from references accepted as credible by the knowledgeable community.

Logical verification includes checking that the code follows the flow diagrams in the CMS. It also involves examining all lines of code on a subroutine-by-subroutine basis. For each subroutine, the following steps are performed:

5. Ensure that subroutine input and output reflect those specified by the CMS.
6. Associate sections of code with Design Elements specified in the Design Approach section of the CMS. Design elements are self-contained entities that perform a certain function such as an equation, or a computational or data processing algorithm. Check that the code accurately implements the design element.

7. Check that the subroutine implemented in code matches any applicable flow charts that appear in the CMS.
8. Review any existing Model Deficiency Reports to determine if previously reported errors that apply to this FE have been addressed.

Code verification consists of checking for errors introduced in the coding process. Computer-Aided Software Engineering (CASE) tools would ordinarily aid this process, but they have not been used in this effort at the request of the SMART Project Office. Procedures used include the following:

1. Check whether internal code documentation (prologues, comment lines) are accurate and adequate.
2. Identify potential overflow and underflow conditions.
3. Identify potential array-bound overwrite conditions.
4. Examine conditional structures for logical branch accessibility.
5. Check miscellaneous code quality characteristics (code structure, variable usage, etc.).

Software Testing

The initial step in software testing is to develop a set of software test cases. Some test cases are designed to exercise the code during run time to verify that the design elements implemented in the code are performing the required calculations correctly. The values produced by the code are compared to values produced from the mathematical algorithms by hand or by independent software. Some test cases are designed to verify that system-specific and user-specified input data arrive correctly in a subroutine and affect the design element as expected. Other test cases are designed to determine how any potential conditions for overflow, underflow, array bound violations, and inaccessible code discovered during desk checking affect model execution.

If the test cases listed above have not exercised all lines of code, additional tests are designed to do so. This reduces the chance that logical errors have been overlooked. The flow diagrams in the CMS are assessed for correctness during the effort to execute all logical branches.

The final software test step is to execute the software test cases and analyze and record the results. These tests are performed in three ways: (1) using an off-line driver to execute subsections of the

code, (2) execution of the entire model in debug mode, or (3) execution of the entire model using instrumented code (diagnostic write statements).

2.0.2 MORS Definitions of Verification Activities

The following terms are approved and used by the Military Operations Research Society [A.1-2]:

Verification: Process of determining that a model implementation accurately represents the developer's conceptual description and specifications.

Logical Verification: The identification of a set of assumptions and interactions for which the model correctly produces intended results. Logical verification determines the appropriateness of the model for a particular application. This is accomplished by the model designer or the developer's IV&V agent.

Code Verification: A rigorous audit of the code to ensure proper implementation, accomplished by both the developer and an independent IV&V agent.

Data Verification: Comparison of model input data to the corresponding known real world or best estimate values. This is typically done by the model user.